# Linear Programming
# ISE 536

## Term Project

Utkarsh Gupta

Sanket Kumbhar

# Part A : Optimizing Freight Container Utilization for a Distribution Center

## Problem: **Minimize Number of Containers**

## Constraints:

1. Weight capacity: **45,000 lbs**
2. Volume capacity: **3,600 in³**
3. Pallet capacity: **60 pallets**

Base Assumptions:

- Whole order in Single Container (No splitting)
- Assumed that the given container max volume already takes double stacked into account

# Part A: Optimal Solution

**Algorithm Used**:

- Initial solution with BFD (to estimate the number of containers).
- Gurobi parameters are set to prioritize finding feasible solutions within time limits using techniques like
    a. Branch-and-Bound: For exploring solution space and used internally by Gurobi
    b. Cutting Planes: To tighten constraints (Gurobi self initialized Clique,Cover, MIR and Gomory cuts)
    c. Heuristics: For faster feasibility.
    d. Added exact container count constraint and then decreasing it iteratively rather than vanilla minimize containers (Latter showed worse runtime for our structure)
- Reducing loop to minimize container count by improving subpar filling
    a. Filtered out containers where neither metric (vol, weight, pallet count) was above 99% utilized and re-optimized them

**Challenges Addressed**:

- Runtime efficiency using Gurobi parameters and loop optimization.
- Issues with Column Generation (too slow).

# Why Gurobi?

- Highly optimized for Mixed Integer Problems
- Its advanced algorithms, such as branch-and-bound and cutting planes, can solve such problems efficiently.
- Vast array of parameters to tune for each type of problem
- Gurobi is recognized as one of the fastest solvers for linear and integer programming problems

# Best Fit Decreasing (BFD) as a Warm Start Heuristic

- **Why BFD?**
  - simple and fast greedy heuristic
  - produces reasonably good initial solutions with low container counts.
  - normalized space utilization ensures that larger orders are processed first, leaving smaller orders for later adjustments.
- **Sort orders**: By normalized space utilization (weight, volume, pallets).

$$\text{Normalized Value} = \frac{w_i}{\text{MAX\_WEIGHT}} + \frac{v_i}{\text{MAX\_VOLUME}} + \frac{p_i}{\text{MAX\_PALLETS}}$$

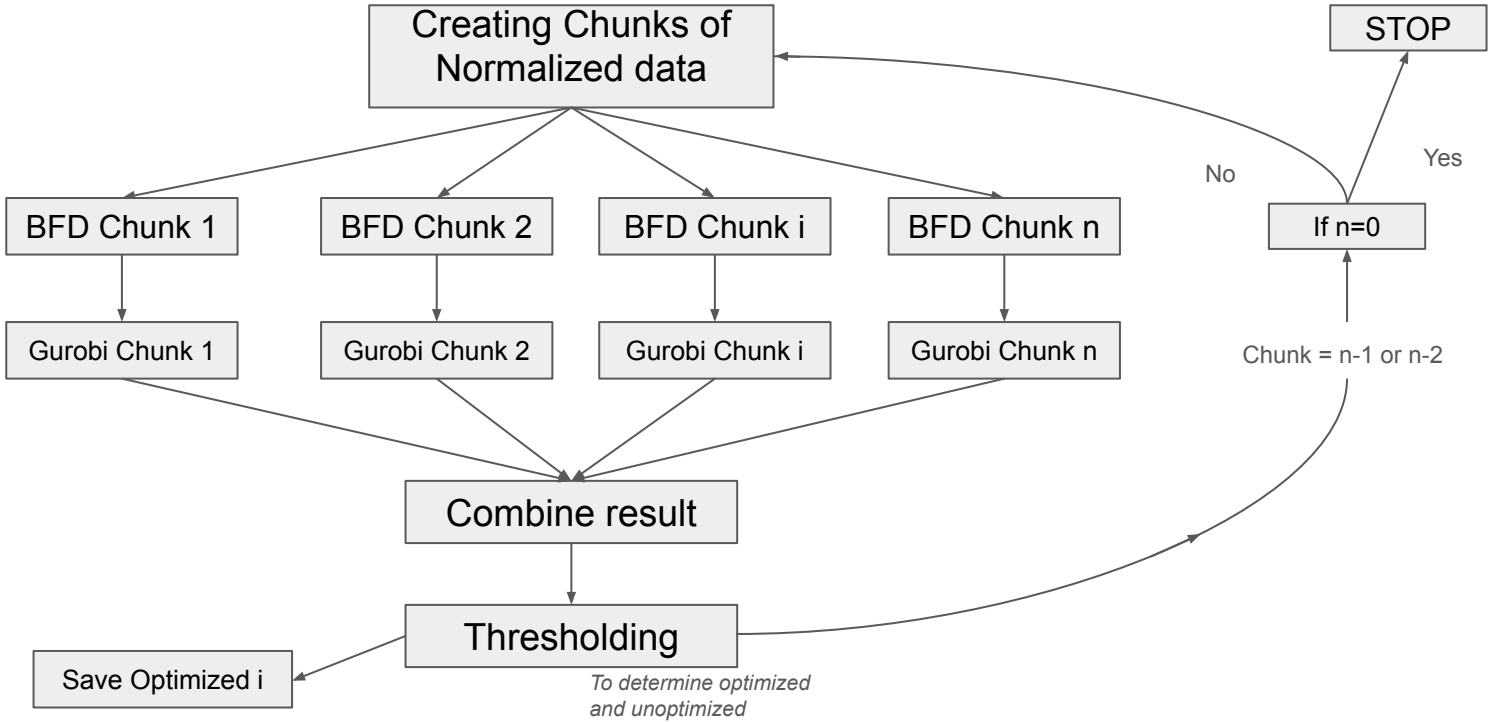
- **Assign Orders**:
  - Fit each order into the "best" container with minimal leftover capacity
  - If no fit is found, create a new container.
- **Properties:** Greedy heuristic, Efficient, Trade-off

# Solution Stats:

| Answer Type | Best Found (Improved Simplex) | Only Heuristic (BFD) |
|---|---|---|
| Container Count | 267 | 273 |
| Avg Weight Cap. Utilization % | 98.02 | 95.44 |
| Avg Volume Cap. Utilization % | 92.70 | 87.84 |
| Avg Pallet Cap. Utilization % | 82.22 | 77.55 |

Theoretical Lower Limit : 261 (due to cumulative weight of the orders)

# Part B: Approximation Solution



# Thresholding Analysis (Part B)

| Threshold(%) | Round 1 | | Round 2 | | Round 3 | Total |
|---|---|---|---|---|---|---|
| | Optimized | Unoptimized | Optimized | Unoptimized | Optimized | |
| 98.8 | 205 | 186(449: Orders) | 84 | 99(229: Orders) | 98 | 387 |
| **99** | **183** | **208(504: Orders)** | **79** | **126(290: Orders)** | **124** | **386** |
| 99.3 | 148 | 243(602: Orders) | 79 | 162(390: Orders) | 160 | 387 |
| 99.5 | 125 | 266(662: Orders) | 75 | 170(420: Orders) | 188 | 388 |

# Divide and Optimize vs. Baseline (BFD)

**Baseline (BFD)**:

- Fast but often inefficient.

**Divide and Optimize**:

- Balances runtime and accuracy.
- Controlled subset sizes limit exponential growth in complexity.

**Runtime Analysis**:

- Divide and Optimize runtime depends on largest subset.
- Trade-off: Precision vs Speed.

# Runtime for Approx Algorithm (and why is it scalable?)

| Dataset | Round 1 | | Round 2 | | Round 3 | | Total (Min) |
|---|---|---|---|---|---|---|---|
| | Chunk Size | Time | Chunk Size | Time | Chunk Size | Time | |
| Part A | 345(3) | ~ 15min | 323(2) | ~14 min | | - | 29Min |
| Part B | 250(4) | ~12min | 254(2) | ~12min | 290 | ~14min | 38Min |

- Processing multiple chunks parallelly
- Runtime mainly depend on chunk size and number of rounds

# Approx algorithm performance comparison (using 1a)

| **Answer Type** | **Best Found (Improved Simplex)** | **Best Approx (Divided Simplex + Filtering)** | **Only Heuristic (BFD)** |
|---|---|---|---|
| Container Count | 267 | 269 | 273 |
| Avg Weight Cap. Utilization % | 98.02 | 97.70 | 95.44 |
| Avg Volume Cap. Utilization % | 92.70 | 92.53 | 87.84 |
| Avg Pallet Cap. Utilization % | 82.22 | 83.21 | 77.55 |

# Approx Solution Stats for 1b:

| **Answer Type** | **Best Approx (Divided Simplex + Filtering)** | **Only Heuristic (BFD)** |
|---|---|---|
| Container Count | **386** | 391 |
| Avg Weight Cap. Utilization % | **97.82** | 96.05 |
| Avg Volume Cap. Utilization % | **95.52** | 89.88 |
| Avg Pallet Cap. Utilization % | **84.81** | 77.32 |

Theoretical Lower Limit : 376  (due to cumulative weight of the orders)

# Analysis of Test Dataset

- Increased Avg volume, weight and pallet count per order in test dataset (~ 45% increase when compared to 1a and 1b)

- Also, the theoretical limiting factor in test dataset is the volume, which was not the case with 1a and 1b

**Effect in optimization:**

The increased average volume, weight, and pallet count per order in the test dataset resulted in higher packing complexity and reduced efficiency in the optimization algorithm. This change likely led to higher utilization of container space but also increased the need for more containers to accommodate the larger orders, impacting the overall cost and performance metrics

**Why did 1a and 1b show similar performance:**

When checking the order stats for 1a and 1b, both showed similar values, hinting as to a reason why our approx. algorithm performed similarly for both

# Our Solution for Test Dataset

| **Answer Type** | **Best Approx (Divided Simplex + Filtering)** | **Only Heuristic (BFD)** |
|---|---|---|
| Container Count | **1139** | 1145 |

| Dataset | Round 1 | | Round 2 | | Round 3 | | Total Time (Min) |
|---|---|---|---|---|---|---|---|
| | Chunk Size | Time | Chunk Size | Time | Chunk Size | Time | |
| Final | 400(5) | ~15min | 346(3) | ~14 min | 480 | ~18min | ~47 |

Theoretical Lower Limit : 1011 (due to cumulative volume of the orders)

# Approx vs BFD - Which one for fast and good solutions?

According to results, it can be concluded that:

- BFD performs very close to optimal for smaller datasets and always provides near-instantaneous solutions for bin packing with a complexity of O(orders), making it an excellent heuristic for obtaining near-optimal solutions. However, it struggles with more complex datasets, leading to an increased gap from the optimal solution.

- Our approximation algorithm consistently delivers a better solution than BFD (as it uses BFD as a baseline). However, the extent of improvement cannot be guaranteed. While it provides a better solution within a reasonable timeframe, its practicality depends on the company's time and logistical constraints—specifically, whether saving on container costs justifies an additional ~30 minutes of computation